CS 134:
Tic Tac Toe (3)
Announcements & Logistics

• **Lab 7** feedback coming soon

• **HW 8** due Monday @ 11 pm

• **Lab 9 Boggle** released today: multi-week partners lab (counts as a two labs in terms of grade; Lab is decomposed into **three** logical parts
  - **Parts 1 & 2 (BoggleLetter & BoggleBoard)** due Wed/Thur 11 pm
  - We will run our tests on these and return automated feedback (similar to Lab 4 part 1), but you are allowed to revise it afterwards

  - **Parts 3 (BoggleGame)** due the following week
  - Please spend time planning and thinking about design before your lab session!

• TA apps due today: [https://csci.williams.edu/tatutor-application/](https://csci.williams.edu/tatutor-application/)

---

**Do You Have Any Questions?**
Last Time

- (Briefly) Looked at important helper methods in the Board class
- Discussed how to build the TTTBoard class
  - Added a grid of TTTLetters to the Board class
  - Discussed logic to check for win on TTTBoard
- Any questions?
Today’s Plan

• Finish our game! Woohoo!
• Implement TTTLetter
  • We already have a good sense of what it should do after our last class, but let’s look at the details
• Implement the game logic
  • Keep track of mouse clicks
  • Keep track of players ("X" and "O" alternate)
• Use methods in TTTLetter and TTTBoard to check for win after each move
TTT Letters

• We have already seen a glimpse of what TTTLetters needs to do
• In fact it has to support this functionality for TTTBoard!

```python
class TTTLetter(builtins.object):
    TTTLetter(win, col=-1, row=-1, letter='')

    A TTT letter has several attributes that define it:
    * _row, _col coordinates indicate its position in the grid (ints)
    * _textObj denotes the Text object from the graphics module,
      which has attributes such as size, style, color, etc
      and supports methods such as getText(), setText() etc.

    Methods defined here:

    __init__(self, win, col=-1, row=-1, letter='')
    Initialize self. See help(type(self)) for accurate signature.

    __repr__(self)
    Return repr(self).

    __str__(self)
    Return str(self).

    getLetter(self)
    Returns letter (text of type str) associated with property textObj

    setLetter(self, char)
```
TTTLetter: __init__

- Let's think about __init__ first

- A TTTLetter is just a “wrapper” around a Text object

- Using passed in parameters (col, row, letter), initialize __slots__ attributes

```python
from graphics import *

class TTTLetter:
    __slots__ = ['_row', '_col', '_textObj']

    def __init__(self, win, col=-1, row=-1, letter=' '):

        # global variables needed for graphical testing
        xInset = 50; yInset = 50; size = 50

        # set row and column attributes
        self._col = col
        self._row = row

        self._textObj = Text(Point(xInset + size * col + size / 2,
                                      yInset + size * row + size / 2), letter)

        self._textObj.setSize(20)
        self._textObj.setStyle("bold")
        self._textObj.setTextColor("black")
        self._textObj.draw(win)
```
Now let’s implement the necessary getter/setter methods

- We don’t need/want to expose the Text object
- We don’t want to allow the row, col to be changed
- We only expose the string (letter) of the Text object, so they are the only getter/setter methods we need
- __str__ useful for debugging and testing

```python
def getLetter(self):
    """Returns letter (text of type str) associated with property textObj""
    return self._textObj.getText()

def setLetter(self, char):
    self._textObj.setText(char)

def __str__(self):
    l, col, row = self.getLetter(), self._col, self._row
    return "{} at Board position ({{}}, {{}})".format(l, col, row)
```
Testing TTTLetter

- It’s always a good idea to test our class and methods in isolation
- Note: No board involved!

```python
win = GraphWin("Tic Tac Toe", 400, 400)
letter = TTTLetter(win, 1, 1, "X")
letter2 = TTTLetter(win, 1, 2, "O")
letter3 = TTTLetter(win, 2, 1, "X")

letter2.setLetter("T")
print(letter2)

# pause and wait for mouse click
# this keeps the window open
point = win.getMouse()

T at Board position (1, 2)
```
Finally... TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages

Let’s think about the “common” case: a valid move in the middle of the game
Finally…TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages.

Now let’s consider the case of a win, draw, or invalid move.
Finally... TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages

Now’s let suppose a player chooses reset
Finally…TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages.

Now’s let suppose a player chooses exit.
Finally… TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages

Finally, let’s handle the click that may be outside of any of the “valid” regions
Finally…TTT Game Logic

- Let’s create a TTT flowchart to help us think through the state of the game at various stages.

Start \rightarrow Wait for mouse click \rightarrow Grid? \rightarrow Empty space? \rightarrow Win? \rightarrow Draw? \rightarrow Exit? \rightarrow Reset state

- Reset state
- Change players
- End
Translating our Logic to Code

- Let's think about __init__:
  - What do we need?
    - a board, player, and maybe numMoves (to detect draws easily)

```python
from graphics import GraphWin
from tttboard import TTTBoard
from tttletter import TTTLetter

class TTTGame:
    __slots__ = ["_board", "_numMoves", "_player"]

def __init__(self, win):
    self._board = TTTBoard(win)
    self._numMoves = 0
    self._player = "X"
```
Translating our Logic to Code

- Now let's write a method for handing a single mouse click (point)
- We need a few if-elif-else checks to handle the grid/reset/exit check
- Let's start with that logic and fill the rest in later

```python
def doOneClick(self, point):
    
    Implements the logic for processing one click. Returns True if play should continue, and False if the game is over.

    # step 1: check for exit button and # exit (return False)
    if self._board.inExit(point):
        
    # step 2: check for reset button and # reset state
    elif self._board.inReset(point):
        
    # step 3: check if click is on a cell # in the grid
    elif self._board.inGrid(point):=
```
Translating our Logic to Code

- Let's handle the “exit” button first (since it's the easiest)

```python
# step 1: check for exit button and
# exit (return False)
if self._board.inExit(point):
    # game over
    return False
```
Translating our Logic to Code

- Now let's handle reset

```python
# step 2: check for reset button and reset game
elif self._board.inReset(point):
    self._board.reset()
    self._board.clearUpperText()
    self._numMoves = 0
    self._player = "X"
```
Translating our Logic to Code

• Finally, let’s handle a “normal” move. Start by getting position and TTTLetter

```python
# step 3: check if click is on a cell
# in the grid
elif self._board.inGrid(point):
    tlet = self._board.getTTTLetterAtPoint(point)
```
Translating our Logic to Code

The rest of our code checks for a valid move, a win, a draw, and updates state accordingly.

At the end, if the move was valid, we swap players.

```python
# make sure this square is vacant
if tlet.getLetter() == "":
    tlet.setLetter(self._player)

# valid move, so increment numMoves
self._numMoves += 1

# check for win or draw
winFlag = self._board.checkForWin(self._player)
if winFlag:
    self._board.setStringToUpperText(self._player + " WINS!")
elif self._numMoves == 9:
    self._board.setStringToUpperText("DRAW!")
else:
    # set player to X or 0
    if self._player == "X":
        self._player = "0"
    else:
        self._player = "X"
```
TTT Summary

• Basic strategy
  • **Board**: start general, don’t think about game specific details
  • **TTTBoard**: extend generic board with TTT specific features
    • Inherit everything, overwrite attributes/methods as needed
  • **TTTLetter**: isolate functionality of a single **TTTLetter** on board
    • Think about what features are necessary/helpful in other classes as well
  • **TTTGame**: think through logic conceptually before writing any code
    • Translate logic into code carefully, testing along the way
Boggle Strategies

• At a high level, Tic Tac Toe and Boggle have a lot in common, but the game state of Boggle is more complicated

• In Lab 9 you should follow a similar strategy to what we did with TTT

• Don’t forget the bigger picture as you implement individual methods

• Think holistically about how the objects/classes work together

• Isolate functionality and test often (use __str__ to print values as needed)

• Discuss logic with partner before writing any code

• Worry about common cases first, but don’t forget the “edge” cases

• Come see instructors/TAs for clarification

GOOD LUCK and HAVE FUN!